18. Matrici

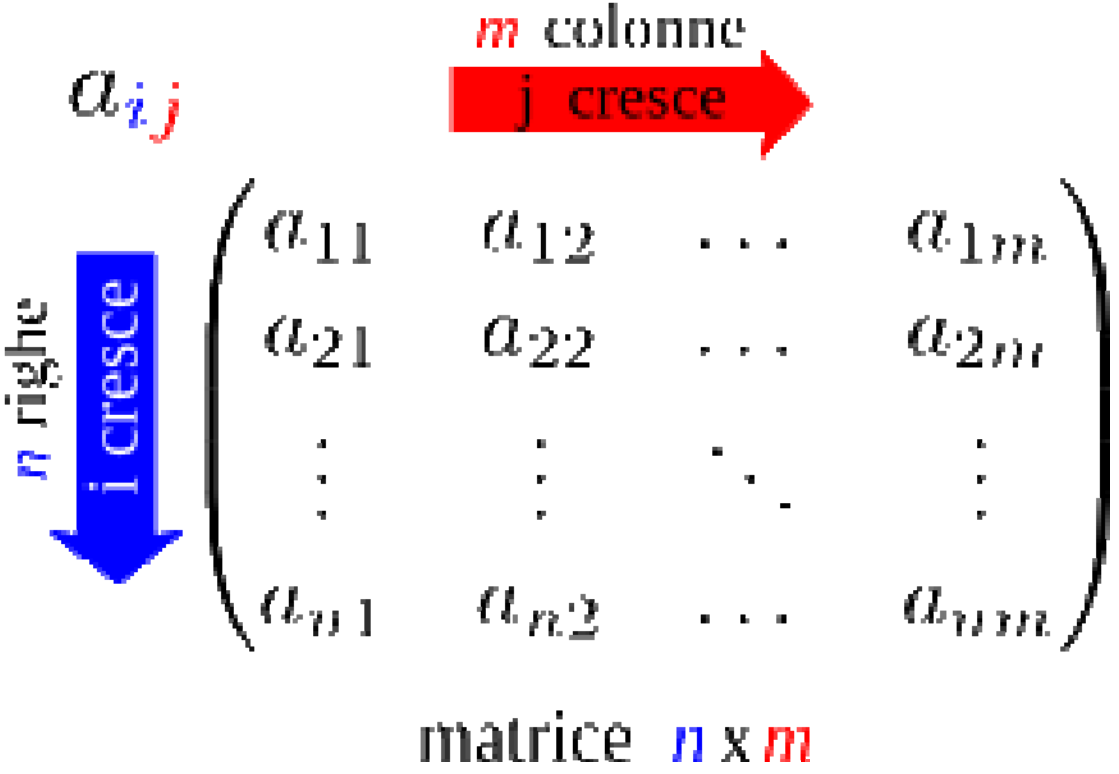
Andrea Marongiu
(andrea.marongiu@unimore.it)
Paolo Valente

Contenuto lezione

- Stringhe
- Struct
  - Operazioni
  - Progettazione strutture dati
  - e passaggio parametri
- Matrici statiche
  - Implementazione
  - Passaggio alle funzioni

Definizione matrice

- Tabella ordinata di elementi
- Esempio bidimensionale:



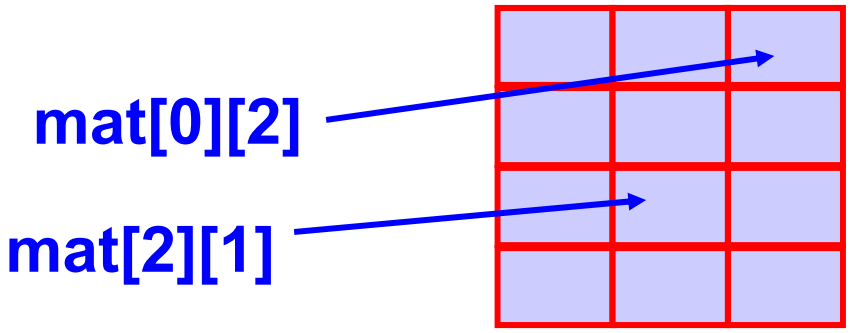
Matrice bidimens. in C/C++

- SINTASSI della definizione di una variabile o di una costante con nome di tipo **matrice bidimensionale statica**:

[const] *<tipo_elementi_matrice>*
<identificatore> **[***<espr-costante>***] [***<espr-costante>***] ;**

Esempio: matrice di 4x3 oggetti (tutti) di tipo **double**

```
double mat[4][3] ;
```



Esercizio 1

- Scrivere un programma che definisca una matrice bidimensionale di interi, di dimensioni stabilite a tempo di scrittura del programma, ed assegni ad ogni elemento un valore letto da stdin. Infine stampi il contenuto della matrice.

Esercizio 1

```
int main(){
  const int M = 3, N = 2 ;
  int mat[M][N] ;
  cout<<"Inserire i "<<M*N<<" elementi della matrice:"<<endl ;

  for (int i = 0 ; i < M ; i++)
    for (int j = 0 ; j < N ; j++)
      cin>>mat[i][j] ;

  cout<<"Contenuto della matrice: "<<endl ;

  for (int i = 0 ; i < M ; i++)
  {
    for (int j = 0 ; j < N ; j++)
      cout<<mat[i][j]<<"\t" ;
    cout<<endl ;  // vai a capo con la nuova riga della matrice
  }

  return 0 ;
}
```

Sintassi matrice k dimensioni

- SINTASSI della definizione di una variabile o di una costante con nome di tipo **matrice statica k-dimensionale**:

[const] *<tipo_elementi_matrice>*
<identificatore> **[***<espr-cost_1>***] [***<espr-cost_2>***] ... [***<espr-cost_K>***] ;**

- *<espr-cost_i>* è il numero di elementi dell'*i*-esima dimensione

- Per accedere ad un elemento bisogna fornire tanti indici quante sono le dimensioni
  - l'*i*-esimo indice assume valori fra 0 e (*<espr-cost_i>* - 1)

- il generico elemento di una matrice è denotato dal nome della matrice seguito dai valori degli indici racchiusi tra [ ]

Inizializzazione matrici

- Generalizzazione della sintassi per gli *array* monodimensionali

Esempio:

```
int mat[3][4] = { {2, 4, 1, 3},
                  {5, 3, 4, 7},
                  {2, 2, 1, 1} } ;
```

- Il numero di colonne **deve** essere specificato
- Il numero di righe può essere omesso, nel qual caso coincide col numero di righe che si inizializzano
- Elementi non inizializzati hanno valori casuali o nulli a seconda che si tratti di un oggetto locale o globale
- Non si possono inizializzare più elementi di quelli presenti

Esercizio 2

- Data una matrice di dimensione *M*x*M* di valori reali inizializzata a tempo di scrittura del programma, si calcoli la differenza tra la somma degli elementi della diagonale principale e la somma degli elementi della diagonale secondaria

- Esempio e suggerimenti nelle prossime slide

Esercizio 2 - Esempio indici matrice 5x5

0,0	0,1	0,2	0,3	0,4
1,0	1,1	1,2	1,3	1,4
2,0	2,1	2,2	2,3	2,4
3,0	3,1	3,2	3,3	3,4
4,0	4,1	4,2	4,3	4,4

Esercizio 2 - Idea

- Gli elementi della diagonale principale sono caratterizzati dagli indici: [i][i]
- Gli elementi della diagonale secondaria sono caratterizzati dagli indici: [i][M-1-i]
- Quindi, per scandire tutti gli elementi delle due diagonali è sufficiente un unico ciclo (e quindi un solo indice)

Esercizio 2 - Algoritmo

- Inizializzare due variabili a 0 e sommarvi rispettivamente i valori degli elementi della prima e della seconda diagonale
- Stampare il valore finale della variabile che contiene la differenza tra le due variabili

Esercizio 2 - Struttura dati

- Una costante (int) per denotare la dimensione della matrice: M=100
- Una matrice bidimensionale di double pari a M x M
- Un indice (int) per scandire la matrice
- Due variabili ausiliarie (double) per sommarvi i valori delle diagonali

Esercizio 2 - Programma

```
main() {
  const int M=100 ;
  double somma_d1=0., somma_d2=0.;
  double mat[M][M];

  for (int i=0; i<M; i++)
    for(int j=0; j<M; j++)
      cin>>mat[i][j] ;

  for (int i=0; i<M; i++)  {
    somma_d1 = somma_d1+mat[i][i];
    somma_d2 = somma_d2+mat[i][M-1-i];
  }
  cout<<“Differenza valori ”
      <<somma_d1-somma_d2<<endl;
}
```

Esercizio 3

- Data una matrice di dimensione *M*x*N* di valori interi, si calcoli il numero complessivo di elementi positivi, negativi e nulli

Esercizio 3 - Algoritmo

- Per scandire tutti gli elementi della matrice possiamo utilizzare due cicli innestati
- Inizializzare due variabili a 0 e sommarvi tutti gli elementi che risultano positivi e negativi
- Serve un'altra variabile per gli elementi nulli?
- Stampa il valore finale delle due o tre variabili

Esercizio 3 - Struttura dati

- Due costanti (int) per denotare la dimensione massima delle righe e delle colonne della matrice: max_R=100, max_C=1000
- Una matrice bidimensionale di int pari a max_R * max_C
- Due (o tre ?) variabili ausiliarie (int) come contatori dei valori positivi e negativi

Esercizio 3 - Programma

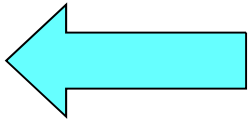
```
main()
 {
    const int max_R = 100, max_C = 1000 ;
    int positivi=0, negativi=0 ;
    int mat[max_R][max_C];

    <si ipotizza che la matrice venga inizializzata in qualche modo>

    for (int i=0; i<max_R; i++)
        { for (int j=0; j<max_C; j++) {
            if (mat[i][j]>0)  positivi++;
            else if (mat[i][j]<0)  negativi++;
       }
    }

  cout<<"Valori positivi= "<<positivi<<", negativi = "<<negativi
      <<", nulli = "<< (max_R*max_C – positivi – negativi)
      <<endl ;
 }
```

Contenuto lezione

- Stringhe
- Struct
  - Operazioni
  - Progettazione strutture dati
  - e passaggio parametri
- Matrici statiche
  - Implementazione 
  - Passaggio alle funzioni

Implementazione matrice

- Considerando la notazione con cui viene definita una matrice e quella con cui si accede ai suoi elementi, forse una matrice è un tipo derivato costruito a partire da un tipo che conosciamo già?

Array di array

- Sì
- Nel linguaggio C/C++, una matrice è a tutti gli effetti un **array di array**
  - Gli array combinati per ottenere una matrice sono organizzati per righe consecutive

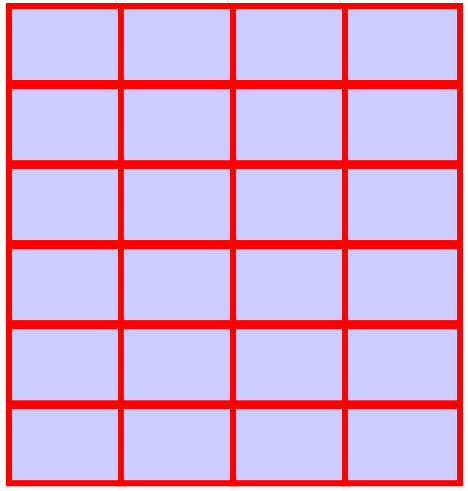
Ad esempio

```
int mat[M][N] ;
```

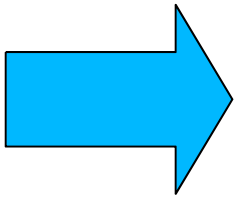
- definisce un array di M array da N elementi ciascuno, ossia M righe da N colonne ciascuna, come mostrato nel seguente esempio numerico

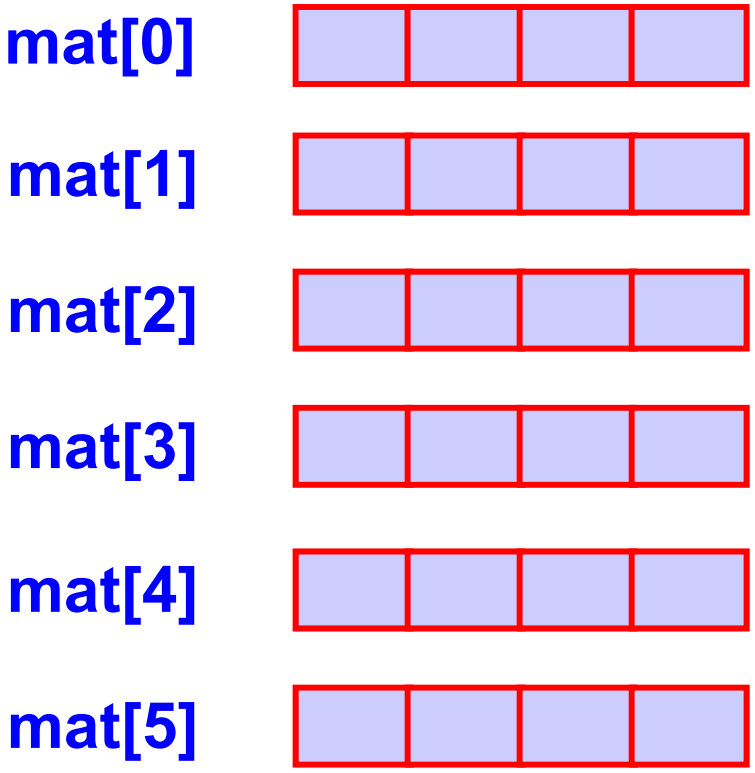
Organizzazione matrice

```
int mat[6][4] ;
```



mat[6][4]

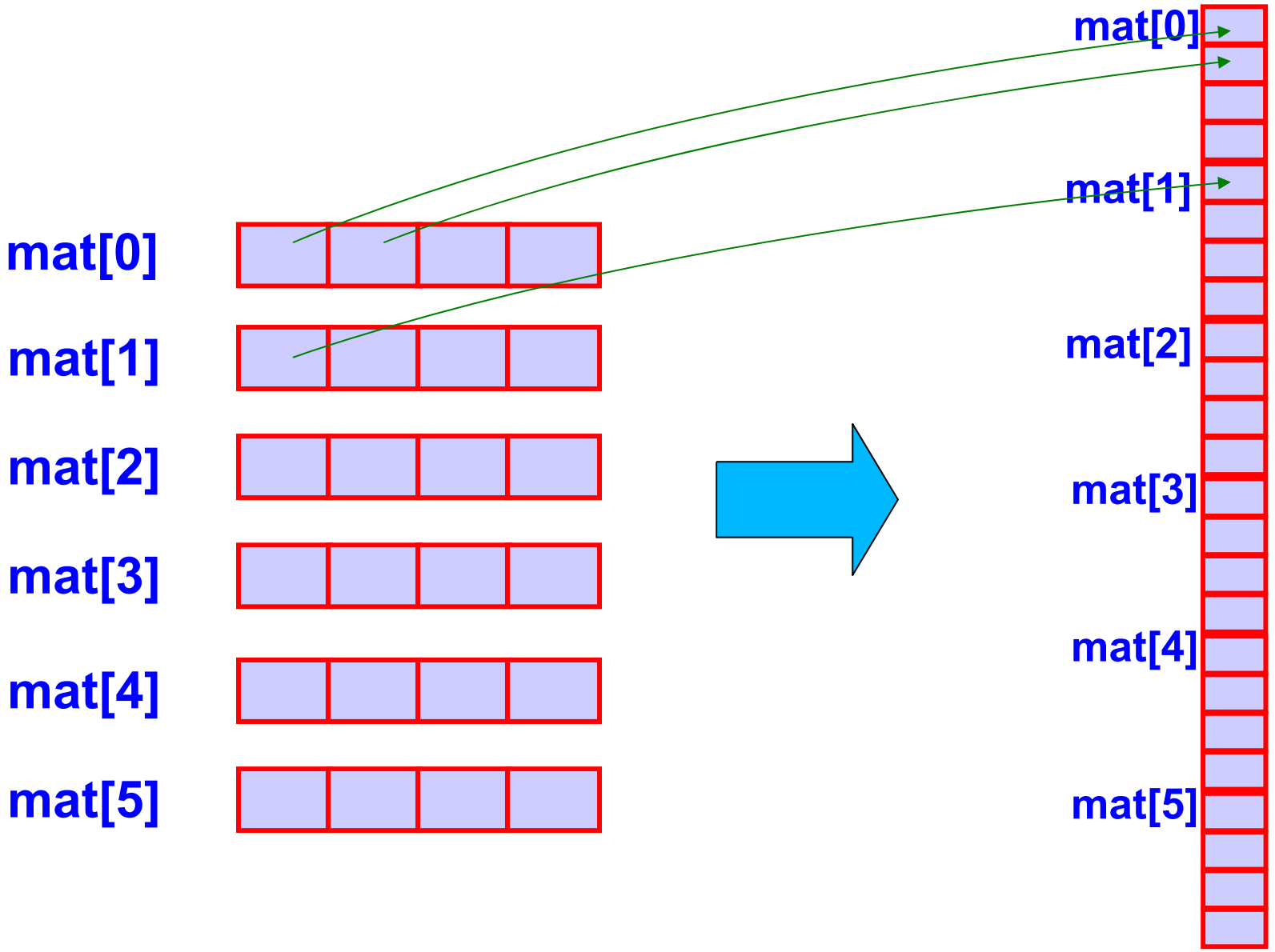




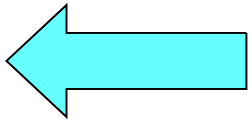
Array di array in memoria 1/2

- Siccome un array è una sequenza contigua di elementi in memoria, allora un array di array è una sequenza contigua di array in memoria
- Un esempio è mostrato nella seguente slide

Array di array in memoria 2/2



Contenuto lezione

- Stringhe
- Struct
  - Operazioni
  - Progettazione strutture dati
  - e passaggio parametri
- Matrici statiche
  - Implementazione
  - Passaggio alle funzioni 

Passaggio righe matrice 2D 1/2

- Riassumendo quanto detto nelle precedenti slide:
- **`int mat[M][N] ;`**
- definisce un array di *M* array da *N* elementi ciascuno
- Cos'è quindi **`mat[i]`** con i = 0, 1, ..., M -1 ?

Passaggio righe matrice 2D 2/2

- E' un *array* di *N* elementi
- Quindi data una matrice di *N* colonne, come si passa una delle righe ad una funzione che prende in ingresso un *array* lunghezza *N*?
- Vediamo con un esercizio:

Esercizio 4 - Passaggio righe matrice 2D

- Scrivere un programma che definisca una matrice bidimensionale di interi di dimensioni stabilite a tempo di scrittura del programma e la inizializzi con valori letti da stdin.

- Successivamente stampi la matrice. Infine calcoli e stampi il valore della somma degli elementi di ciascuna riga.

- La somma degli elementi di ciascuna riga va effettuata mediante una funzione che prende in ingresso un array di interi e ritorna il valore della somma degli elementi.
  - *La funzione non legge nulla da stdin e non scrive nulla su stdout.*

Esercizio 4 - Passaggio righe matrice 2D

```
int calcola_somma (const int a[], int n){
  int somma = 0 ;
  for (int i = 0 ; i < n ; i++)
    somma += a[i] ;
  return somma ;
}

int main(){
  const int M = 3, N = 2; int mat[M][N];

  cout<<"Inserire i "<<M*N<<" elementi della matrice:"<<endl ;
  for (int i = 0 ; i < M ; i++)
    for (int j = 0 ; j < N ; j++)
      cin>>mat[i][j] ;

  cout<<"Contenuto della matrice: "<<endl ;
  for (int i = 0 ; i < M ; i++) {
    for (int j = 0 ; j < N ; j++)
      cout<<mat[i][j]<<"\t" ;
    cout<<endl ;
  }
  cout<<endl ;
  for (int i = 0 ; i < M ; i++)
    cout<<calcola_somma(mat[i], N)<<endl ;

  return 0 ;
}
```

Passaggio righe matrice 2D 2/2

- ESERCIZIO PER CASA
- Generalizzazione per il passaggio di fette di matrici con più di due dimensioni

Passaggio matrici

- Così come gli array monodimensionali, gli array di array sono **passati per riferimento**
- La dichiarazione/definizione di un parametro formale di tipo matrice bidimensionale è la seguente:
- [**const**] *<tipo_elementi>*
- *<identificatore>* **[][***<numero_colonne>***]**
  - Nessuna indicazione del numero di righe !!!
  - La funzione pertanto non conosce implicitamente il numero di righe della matrice
  - Se presente, il qualificatore **const** fa sì che la matrice non sia modificabile
- Nell'invocazione della funzione, una matrice si passa **scrivendone semplicemente il nome**

Esempio

```
const int num_col = 4 ;

void fun (int mat[][num_col], int num_righe) ;

main()
{
   const int M = 3 ;
   int A[M][num_col] ;
   fun(A, M) ;
   ...
}
```

Matrici e indirizzi

- Di quale informazione ha bisogno il compilatore per poter generare il codice che accede al generico elemento di una matrice?
  - Dell'indirizzo di tale elemento in memoria

- Di quali informazioni ha bisogno per calcolare tale indirizzo?
  - Ricordate che le righe sono memorizzate l'una dopo l'altra

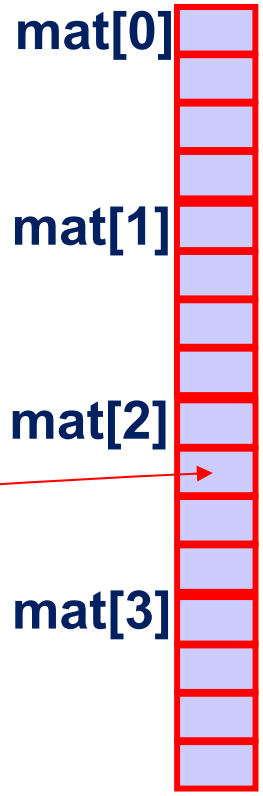
Matrici e indirizzi

1) La prima locazione in cui è memorizzata la matrice
2) Le dimensioni di ciascun elemento (numero di byte occupate, dipende dal tipo degli elementi)
3) La lunghezza di ciascuna riga, ossia il numero di colonne della matrice

- Tali informazioni servono per accedere, ad esempio, al secondo elemento della terza riga di questa matrice, ossia l'elemento di indici [2][1]

- L'indirizzo dell'elemento è infatti dato da:

indirizzo_matrice + dim_elementi(lun_riga*2 + 1)*



Indirizzo generico elemento

- In generale l'indirizzo del generico elemento di indici *i* e *j* è dato da

*indirizzo_matrice + dim_elementi * (lun_riga * i + j)*

- Alla luce di quanto abbiamo appena capito, come mai è obbligatorio passare il numero di colonne?
  - *Perché, come visto, è necessario conoscere il numero di colonne di una matrice bidimensionale per calcolare l'indirizzo di un suo generico elemento*

Array di stringhe

Per analogia con quanto detto in precedenza, un array di stringhe si realizza mediante una matrice di tipo **`char`**

Esempio: Elenco dei nomi dei giorni della settimana:

```
char giorni[7][11] =
{ “lunedi'”, “martedi'”, “mercoledi'”,
  “giovedi'”, “venerdi'”, “sabato”,
  “domenica” } ;
```

l	u	n	e	d	i	'	\0			
m	a	r	t	e	d	i	'	\0		
m	e	r	c	o	l	e	d	i	'	\0
g	i	o	v	e	d	i	'	\0		
v	e	n	e	r	d	i	'	\0		
s	a	b	a	t	o	\0				
d	o	m	e	n	i	c	a	\0		

Esercizio 5

- Scrivere una funzione

```
bool quadratomagico (int q[][N]);
```

- che verifica se una matrice quadrata di interi di ordine N e' un quadrato magico, ovvero se la somma degli elementi di ogni riga, di ogni colonna e delle diagonali è identica. La funzione ritorna "1" se la matrice è un quadrato magico, "0" altrimenti.

- Esempio di quadrato magico 3x3:
  2 9 4
  7 5 3
  6 1 8

- scrivere anche un semplice main() che: a) definisce una matrice NxN; b) richiede l'immissione da tastiera degli elementi della matrice; c) richiama la funzione, passandole come parametro la matrice; d) stampa il risultato della computazione.

Esercizio 5

```
const int N = 3;
bool quadratomagico (int q[][N]) {
  int s, somma = 0;          // Inizializzazione di somma
  for (int j=0; j<N; j++)
    somma += q[0][j];

  for (int i=1; i<N; i++) {// Verifica righe
    s = 0;
    for (int j=0; j<N; j++)
      s += q[i][j];
    if (s != somma) return false;
  }
  for (int j=0; j<N; j++) {// Verifica colonne
    s = 0;
    for (int i=0; i<N; i++)
      s += q[i][j];
    if (s != somma) return false ;
  }
  // Verifica diagonale principale
  s = 0;
  for (int i=0; i<N; i++)
    s += q[i][i];
  if (s != somma) return false ;
  //Verifica diagonale secondaria
  s = 0;
  for (int i=0; i<N; i++)
    s += q[i][N-1-i];
  if (s != somma) return false ;

  return true ;
}
```

```
void stampa (int mat[][N]){
  for (int i=0; i<N; i++) {
    cout<<endl ;
    for (int j=0; j<N; j++)
      cout<<mat[i][j]<<"\t" ;
  }
  cout<<endl ;
}

main() {
  int m[N][N] ;
  cout<< "Inizializzazione del
quadrato "<<N<<"x"<<N<<":\n" ;

  for (int i=0; i<N; i++)
    for (int j=0; j<N; j++)
      cin>>m[i][j] ;

  cout<< "\nIl quadrato:"<<endl ;
  stampa(m);
  cout<<(quadratomagico(m) ? "\n"
: "\nnon ")<<"e' un quadrato
magico."<<endl ;
}
```

Esercizio 6

- Scrivere un programma che chieda all'utente di inserire il valore degli elementi di due matrici, in generale non quadrate, e di dimensioni predefinite (a tempo di scrittura del programma), effettui la somma di tali matrici e stampi la matrice risultante.

- L'inserimento dei valori in ciascuna delle due matrici deve essere effettuato mediante una funzione che prenda in ingresso la matrice (una sola) da inizializzare e tutte le eventuali altre informazioni necessarie.

- La stampa dei valori di ciascuna delle due matrici deve essere effettuata mediante una funzione che prenda in ingresso una (sola) matrice e tutte le eventuali altre informazioni necessarie.

- La somma deve essere effettuata da una funzione che prende in ingresso due matrici e tutte le eventuali altre informazioni necessarie, e mette il risultato in una terza matrice.

Esercizio 6

```
const int N = 2;

void stampa (int mat[][N], int m)
{
  for (int i=0; i<m; i++) {
    for (int j=0; j<N; j++)
      cout<<mat[i][j]<<"\t" ;
    cout<<endl ;
  }
}

void riempi (int mat[][N], int m)
{
  for (int i=0; i<m; i++) {
    for (int j=0; j<N; j++)
      cout<<"Inserisci valore di riga «
          <<i+1<<" colonna "<<j+1<<":\n" ;
      cin>>mat[i][j] ;
  }
}

void somma_matrici (const int A[][NUM_C],
                    const int B[][NUM_C],
                    int ris[][NUM_C],
                    int m)
{
  for (int i=0; i<m; i++) {
    for (int j=0; j<N; j++)
      ris[i][j]=A[i][j] + B[i][j];
}
```

```
main() {
  const int NUM_R = 3 ;
  int mat1[NUM_R][NUM_C],
      mat2[NUM_R][NUM_C],
      mats[NUM_R][NUM_C];

cout<<"Caricamento matrice 1\n" ;
riempi_matrice(mat1, NUM_R) ;

cout<<"Caricamento matrice 2\n" ;
riempi_matrice(mat2, NUM_R) ;

/*Visualizzazione*/
cout<<"Matrice 1:\n" ;
stampa_matrice(mat1, NUM_R) ;
cout<<"Matrice 2:\n" ;
stampa_matrice(mat2, NUM_R) ;

/*Calcolo della matrice somma*/
somma_matrici(mat1, mat2, mats, NUM_R) ;
/*Visualizzazione della matrice somma*/
cout<<"Matrice somma:\n" ;
stampa_matrice(mats, NUM_R) ;

return 0;
}
```

Esercizio 7

- Data una matrice quadrata A di dimensione M (M righe ed M colonne) (sia aij l'elemento individuato dalla riga i-esima e dalla colonna j-esima), si dice trasposta di A una matrice quadrata B di dimensione M (sia bnm l'elemento individuato dalla riga N-esima e dalla colonna M-esima), tale che aij == bji ove 1 <= i <= M e 1 <= j <= M.
- In sostanza la trasposta fa corrispondere alla prima riga di A, la prima colonna di B, alla seconda riga di A la seconda colonna di B e così via. (Gli indici dei coefficienti utilizzati nella definizione sono ovviamente quelli 'matematici': considerando, ad esempio, la prima riga, il primo elemento ha indice 1 e l'ultimo ha indice N).

- Ad esempio, data la matrice

1 4 2		1 3 8
3 6 7	la trasposta è	4 6 2
8 2 5		2 7 5

- Scrivere un programma che, data una matrice (quadrata) di interi, la stampa, ne calcola la trasposta e stampa la nuova matrice così ottenuta.
- La matrice di partenza va definita a tempo di scrittura del programma.
- Sia il calcolo della trasposta che la stampa devono essere implementate, rispettivamente, in due funzioni dedicate.

Esercizio 7

- ALGORITMO per il calcolo della matrice trasposta

- Quando i == 0, scorro la prima riga e ne scambio gli elementi con quelli della prima colonna.
- Quando i>1, lavoro sulla sottomatrice che parte dalla colonna e dalla riga di indice i, e di nuovo scambio riga e colonna.
- Lavoro sull'array contenitore (quadrato) che contiene la matrice, ottenendo implicitamente lo scambio riga/colonna della matrice contenuta.
- Nel ciclo più interno parto da j=i+1 perché sarebbe inutile scambiare il primo elemento della (sottomatrice) con se stesso

Esercizio 7

```
const int M = 3;

void stampa (int mat[][N], int m)
{
  for (int i=0; i<m; i++) {
    for (int j=0; j<N; j++)
      cout<<mat[i][j]<<"\t" ;
    cout<<endl ;
  }
}

void trasposta(int m[][M])
{
  for (int i = 0 ; i < M ; i++)
    for (int j = i + 1 ; j < M ; j++)
    {
      m[i][j] -= m[j][i] ; // a == x - y
      m[j][i] += m[i][j] ; // b == x
      // a == -(x-y-x) == -(-y) == y
      m[i][j] = -(m[i][j]-m[j][i]) ;
    }
}
```

```
main() {
  int m[M][M] = { {1, 2, 8},
                  {4, 5, 3},
                  {7, 8, 4} };

  cout<<"Matrice iniziale:\n" ;
  // stampa della matrice originaria
  stampa_matrice(m, M);
  // calcolo della matrice trasposta
  trasposta(m);
  cout<<"Dopo la 1 trasposizione:\n" ;
  // stampa della matrice trasposta
  // con una nuova trasposizione
  // riottengo la matrice originaria
  stampa_matrice(m, M);
  cout<<"Dopo la 2 trasposizione:\n" ;
  trasposta(m) ;

  // stampa della matrice originaria
  stampa_matrice(m, M) ;
}
```

Esercizio 8 - Battaglia navale semplificata

- Realizzare un programma che, dopo aver fatto creare una mappa 10x10 con 12 navi da 1 cella in posizioni casuali, consenta ad un giocatore di “scoprire” tutte le posizioni delle nave avversaria.
- La classifica dei record viene mantenuta rispetto al numero dei colpi necessari per scoprire tutte le nave nemiche.
- Estensione: si visualizzi la mappa, con la posizione delle navi scoperte, i tiri effettuati andati a vuoto, e quelli andati a buon fine
- Per implementare bene il programma partire dalla realizzazione delle seguenti funzioni propedeutiche

Esercizio 8 - Battaglia navale semplificata

FUNZIONI PROPEDEUTICHE

- Scrivere una funzione **INSERT** che riceva in input un numero di navi e le inserisca casualmente in una mappa di dimensioni 10x10
  - Si assuma che ciascuna nave occupi 1 cella
  - Si faccia attenzione a non posizionare le navi in celle coincidenti

- Scrivere una funzione **TIRO** che riceva in input una coordinata (ovvero due elementi interi), e restituisca se il tiro ha colpito o meno una nave

Esercizio 9 - Battaglia navale completa

- Scrivere un programma che
  - crei una mappa con le seguenti navi in posizioni casuali:
    - 1 nave da 4 celle, 2 navi da 3 celle, 3 navi da 2 celle, 4 navi da 1 cella (scegliere a proprio piacimento le dimensioni della mappa)
  - consenta ad un giocatore di “scoprire” le posizioni delle navi avversarie
  - Mantenga una classifica dei record rispetto al numero dei colpi necessari per scoprire tutte le navi nemiche
- Estensione: si visualizzi la mappa, con la posizione delle navi scoperte, i tiri effettuati andati a vuoto, e quelli andati a buon fine
- Per implementare bene il programma partire dalla realizzazione delle seguenti funzioni propedeutiche

Esercizio 9 - Battaglia navale completa

FUNZIONI PROPEDEUTICHE

- Data una mappa di dimensione MxM, si inseriscano casualmente (in posizioni non sovrapposte):
  - 1 nave da 4 celle
  - 2 navi da 3 celle
  - 3 navi da 2 celle
  - 4 navi da 1 cella
- Si accettano navi in diagonale?
- Scrivere poi una funzione che, presa in ingresso una coordinata, stampi su video se il tiro ha colpito o meno una nave

Esercizio 10 – Gioco della vita

- Una mappa di dimensione *N*x*M* rappresenta il mondo. Ogni cella può essere occupata o meno da un organismo. Partendo da una configurazione iniziale di organismi, questa popolazione evolve nel tempo secondo tre regole genetiche:
  - un organismo sopravvive fino alla generazione successiva se ha 2 o 3 vicini;
  - un organismo muore, lasciando la cella vuota, se ha più di 3 o meno di 2 vicini;
  - ogni cella vuota con 3 vicini diventa una cella di nascita e alla generazione successiva viene occupata da un organismo.
- Si visualizzi l'evoluzione della popolazione nel tempo

Esercizio 10 – Gioco della vita

NOTA

- Il concetto di “vicinanza” in una tabella raffigurante il mondo può essere interpretato in 2 modi:
  - Al di là dei bordi c’è il vuoto che non influenza il gioco, per cui ci sono punti interni che hanno 8 potenziali “vicini”, punti sulle righe e colonne estreme che hanno 5 “vicini”, punti ai vertici che hanno 3 “vicini”
  - I bordi estremi confinano tra di loro: la colonna “0” è “vicina” alla colonna “M-1”, così come la riga “0” è “vicina” alla riga “N-1” (con attenzione a trattare i vertici!)